Kalman filtering for object tracking

Meghna P Ayyar and Smriti Joshi

I. INTRODUCTION

This laboratory report presents object tracking by implementing a Kalman Filter to track the objects from a video. The routines have been written in C++ using OpenCV 4.2.0 library. The development platform used are: Linux with the Eclipse IDE and Windows using the Visual Studio Community 2019 IDE. Using the output of OpenCV's background subtraction module that assumes a static camera, foreground segmentation masks have been generated. In the current implementation, blobs are extracted from these masks and only the largest blob is considered for tracking across the video frames. Two other tasks involving the analyses of the implementation with toy videos and real life videos have been performed to show the various considerations required to make the filter work well in varying conditions. The filter has been implemented with two different models namely: constant velocity and constant acceleration. The results and discussion have been evaluated on the given set of video sequences and are summarized in Section V.

II. METHOD

The current task focuses on the tracking only one object across frames in various sequences. For this purpose foreground binary masks have been used along with blob detection and performing Kalman tracking on the center of the blob. The details about each of the methods considered have been discussed in this section. Further details regarding the implementation have been summarized in Section III.

A. *Blob Detection with GrassFire algorithm*

The main task to be performed after the segmentation of the foreground is to identify the major blobs from the given segmentation mask. For this purpose the GrassFire algorithm has been employed. The algorithm does a pixel by pixel search on the given mask to detect the pixels that are connected together and so form an object. Once the blob is detected, the largest blob is selected, its center is obtained and stored for further analysis.

B. *Blob Tracking with Kalman Filter*

The blob tracking has been implemented with Kalman Filter. This filter is known to be the best tracking method in linear systems and has been widely used for many applications. This is due to several reasons: it effectively describes the random nature of experimental measures and it also provides the quality of estimation by providing the variance of estimation error. Kalman filter's are computationally inexpensive because of their recursive structure- real time executions without storing observations or previous estimations. Kalman filter comprise of two stages: Prediction(time update) and Correction(measurement update).

Prediction:

Projection of the state ahead (x_k^-) using the state transition model, A is done by

$$x_k^- = A_k x_{k-1} \tag{1}$$

The updation of error covariance

$$P_k^- = A_k P_{k-1} P_k^T + Q_k \tag{2}$$

Correction:
The correction part of the filter is where the main parameter of the Kalman Gain comes into play which is what makes the filter so useful. The computation of Kalman Gain (K_k) is done by

$$S_k = H_k P_k^- H k^T + R_k \tag{3}$$

$$K_k = P_k^- H k^T (S_k)^{-1} \tag{4}$$

Updating state estimate by measurement z_k

$$x_k = x_k^- + K_k(z_k - H_k x_k^-) \tag{5}$$

Updating Error Covariance

$$P_k = (I - K_k H_k) P_k^-$$

In the current implementation, the kalman filter has been constructed using two model: constant velocity and constant acceleration.

1) Constant Velocity Model: For this model, the state is given as $x = [x, \dot{x}, y, \dot{y}]$ with x, y as the position and $(\dot{x}, doty$ as the velocity of the object being tracked. The transition matrix in this case is defined as follows:

1 1 0 0
0 1 0 0
0 0 1 1
0 0 0 1

From the second and fourth row, it can be observed that velocity along x and y direction, $\dot{x}, \dot{y}$, remains constant under state transition.

2) Constant Acceleration Model: For this model, the state is given as $x = [x, \dot{x}, \ddot{x}, y, \dot{y}, \ddot{y}]$ where the (x, y) parameters are the position, $(\dot{x}, \dot{y})$ parameters are the velocity and $(\ddot{x}, \ddot{y})$ parameters are the accelration of the object being tracked. The transition matrix for this case is defined as follows:

$$\begin{matrix} 1 & 1 & 0.5 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0.5 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

From the third and sixth row, it can be observed that acceleration along x and y direction, $\ddot{x}, \ddot{y}$, remains constant under state transition.

III. IMPLEMENTATION

This section explains in detail the implementation of the methods described in section II. The inputs, outputs and other coding considerations followed to achieve the objectives have been mentioned in detail. In addition, all the following methods use the function PMOG2 from OpenCV to create foreground segmentation masks based on a specified learning rate parameter.

A. *Foreground Segmentation with PMOG2*

The foreground segmentation is obtained through inbuilt function of OpenCV based on mixture of gaussians. The initial parameters set for the detector are: history =50 , varThreshold=16. The history parameter controls the number of frames that are considered in the background model and the variance threshold controls the variance which decides if a pixel belongs to the model or not. The learning rate is another parameter which is varied to adjust the rate at which the background is modelled. These parameters have been varied for each of the test sequences accordingly. A morphological operation using a rectangle element of size (3,3) [default value which is also modified later]is applied on the obtained foreground segmentation mask to remove noise and fill the holes.

B. *Blob Detection with GrassFire algorithm*

The function extractBlobs() is used to extract the blobs from a given foreground mask. The connectivity parameter is given as an argument to this function for the use of the Grassfire algorithm. This parameter decides the number of neighbours to be searched while detecting a connected pixel set. The fill operation of the Grassfire algorithm has been implemented using the floodFill() function from OpenCV. The burn operation of the algorithm has been implemented by using the mask argument of floodFill to get a mask which has 1 in all the pixels that were filled by the function. Using matrix operations the foreground pixels detected as blob are set to 0 (the burn step). A rectangle object is passed as argument to the function which is then used to initialize a new blob.

As the sequences require tracking for a single object, the function then checks for the largest blob amongst the blobs obtained from a frame. This has been done by comparing the current blob area with the previously obtained blob area. It is also important to ignore very small blobs originated due to noise. This is why minimum width and minimum height parameters are used to select blobs with the appropriate sizes. The min_height and min_width parameters are also passed as inputs to the function.

C. *Blob Tracking with Kalman Filter*

For object tracking with Kalman filter, various parameters and functions are used which that have been encapsulated in KalmanFilterOp class. The choice for the model is passed through to the constructor of KalmanFilterOp object through boolean variable called vel_flag. A wrapper class of OpenCV called KalmanFilter has been used for this implementation and its object is taken as the member variable of KalmanFilterOp class. This KalmanFilter object for this class object has been initialised with a function InitialiseKalman() which does the appropriate intialization of the varoius parameters according to the type of model specified by vel_flag. For each detected object, the measurement is obtained as the center of the detected blob. This blob is passed to Start() function of the KalmanFilterOp object from tracking. The first time a blob is detected this initial state of the object is stored as the initial position of the blob. The prediction step is carried out by the ApplyKalmanPrediction() function on the KalmanFilter object. If a measurement is made, then the estimated state is corrected with the correct() function of the KalmanFilter in ApplyKalmanCorrection() method and the position is stored in a list. Otherwise, if there is no measurement, the state obtained from the prediction is stored in the list for that frame. Every measurement, prediction and correction states of the KalmanFilter object are stored in separate lists which are later used for visualizations of the predictions and tracking.

D. *Running the code*

- **Implementation Floodfill (Blob detection)**: To run the blob detection, different learning parameters are used for each given task for best results. The learning rate can be changed in line 58 in Lab3.cpp. The connectivity value can be set as 4 or 8 for extractBlobs() function.
- **Implementation of constant velocity and constant acceleration model** :By setting vel_flag in line 63 of Lab3.cpp, one of the two model can be chosen. true corresponds to constant velocity model and false value corresponds to constant acceleration model. Based on this flag, the appropriate KalmanFilter object is initialised in KalmanFilterOp() constructor. For the video sequences in Task 2 and Task 3, the value of R is changed in line 88 of Kalman.cpp.
- **blobs.cpp**: In addition, the blobs.cpp file contains the extractBlobs() function along with some drawing func-

tion to paint circles and draw trajectories that can be used to visualize the results of the tracker.

IV. DATA

The assignment comprised of three sub-tasks. Each of these sub-tasks have different types of video sequences on which the implementations were tested on.

A. *Task 3.1*

The toy video singleball.mp4 used for this task comprised of the movement of a single ball across the frame. It was occluded by a box towards the end of the frame and then reappeared by passing through the box.

B. *Task 3.2*

Name	**Duration**	**No. Frames**	**Features**
video2.mp4	13s	111	A ball moves across the table in a diagonal. The shape of the ball is blurry
video3.mp4	15s	128	Ball moves diagonally to the wall. Hits it slows down and makes a movement in a V shape.
video5.mp4	14s	114	Ball is dropped vertically. Bounces a few times, and stops. Its speed decrease with each bounce. Camera is parallel to the floor.
video6.mp4	16s	131	Box is placed in the middle. Ball rolls from left side, gets occluded by the box, reappears. Then hits the wall and stops.

TABLE I

VIDEO DETAILS FOR THE TASK 3.2

C. *Task 3.3*

Table II summarizes the features and details regarding the video sequences used for this task.

Name	Duration	No. Frames	Features
abandoned box	16	401	Low resolution frames Cyclist stops on the traffic signal
boats	38	951	Dynamic Background with moving water in the lake Boats turns back later in the sequences
pedestrians	9	226	Highly illuminated frames with shadows Pedestrians stops at periodic intervals
street corner at night	3	90	Low illumination Fast moving car object

TABLE II

VIDEO DETAILS FOR THE TASK 3.3

V. RESULTS & ANALYSIS

A. *TASK 3.1*

On evaluating the algorithm on toy dataset- singleball.mp4, it was observed that both constant velocity model(as shown in Figure 1) and constant acceleration model(as shown in Figure 2) work well with detecting and tracking the ball outside and inside the box. The filter parameters for this sequence are kept at the default values. The learning rate is set to 0.001. The min_width and min_height parameters are set as (10,10) which is sufficient to detect the ball and ignore the small noise in the background for blob detection. For these parameters, we can observe that the path predicted by the Kalman tracker when the ball is occluded is slightly more curved in the acceleration model than in the constant velocity model which has a more straight predicted path.

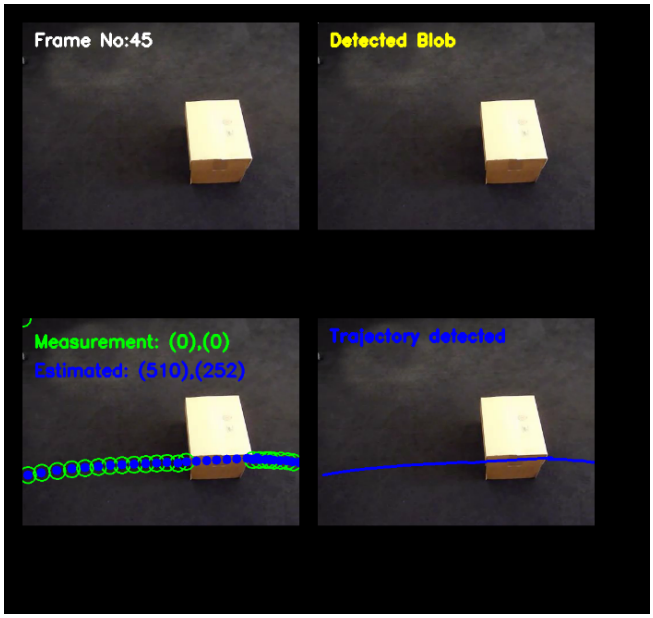

Fig. 1. Results for constant velocity model(singleball.mp4). Blue circles: estimated points, Green circles: measurement points

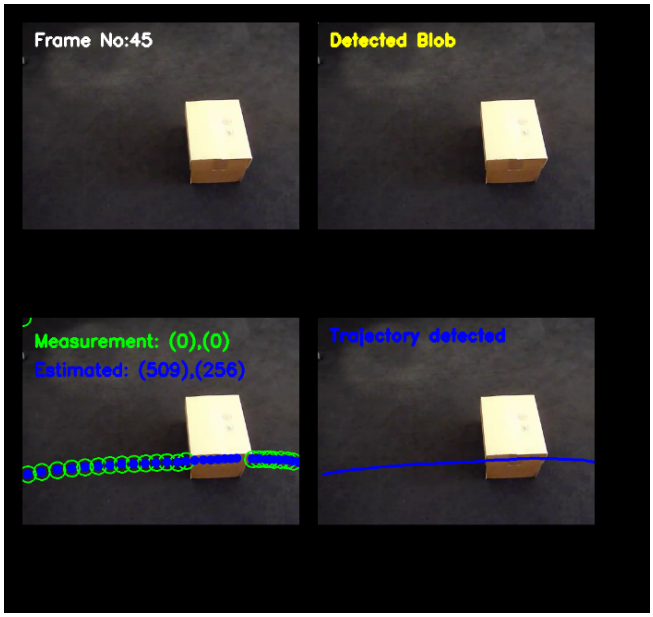

Fig. 2. Results for constant acceleration model(singleball.mp4). Blue circles: estimated points, Green circles: measurement points

B. *TASK 3.2*

The results have been evaluated separately for each of the video sequences of Task 3.2.

Figures 3 and 4 show the results that were obtained for Video2. The parameters were set as : learning rate = 0.001, min_height = 50, min_width = 50, measurement noise covariance, R = [25, 0;0 25]. This video sequence did not have occlusions and had a simple motion of the ball on a static background. In this case not much difference was observed in the behaviour of the two models and both were relatively successful in tracking the motion of the object. Changing the values of R did not produce much change in the predictions due to the motion of the object being much simpler.

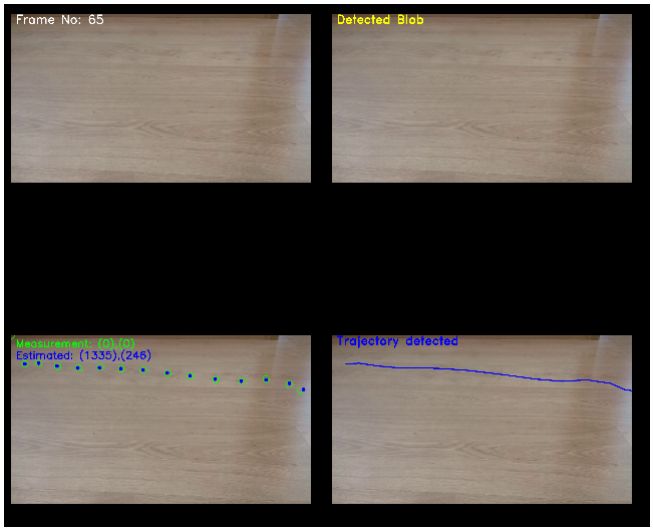

Fig. 3. Results for constant velocity model(Video2.mp4). Blue circles: estimated points, Green circles: measurement points

Fig. 4. Results for constant acceleration model(Video2.mp4). Blue circles: estimated points, Green circles: measurement points

Figure 5 shows the results for Video3. The parameters were set as: learning rate = -1, min_height = 50, min_width = 50. Different values of R(Measurement Noise Covariance) are experimented with: R = [25,0;0,25], [250,0;0,250]. It can be seen from the Figure 5 i that for R = [250,0;0,250], the rebound effect of estimated state is decreased in comparison to R = [25,0;0,25] and better results are observed. This is because increasing the values in R presses the model to trust the measurements more than prediction. The curving of the prediction once the ball disappears in case of the acceleration model is very visible which can be seen reduced when the value of R is increased. In this case the acceleration model produces an effect not much modelling the motion of the ball as it does not match it. The constant velocity model seems to suit this particular motion better as towards the end it predicts a straight line path for the ball once it disappears which is what would have happened in real life (provided there were no other obstacles and before coming to rest). The learning rate is decreased from 0.001 to -1 to adapt for background change. The reason for the choice of this learning rate has been discussed in detail combining the behaviour of this video and Video5.

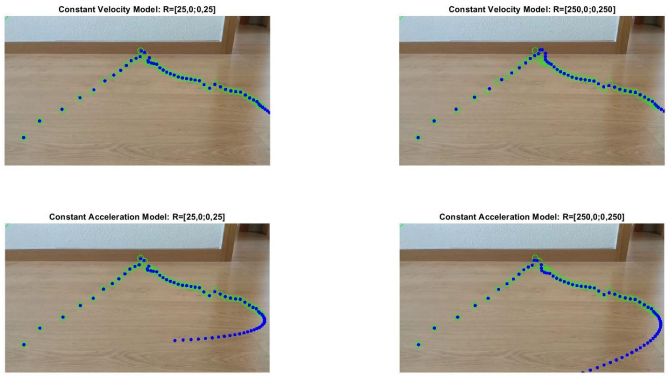

Fig. 5. Comparison of results obtained for constant velocity model and constant acceleration model for different value of measurement noise covariance, R. (Video3.mp4). Blue circles: estimated points, Green circles: measurement points

In Figure 6, the estimated points, predicted points and final trajectories have been shown for video5. The parameters for this sequence are set as: learning rate = -1, min_height = 50, min_width = 50, measurement noise covariance, R = [25,0;0 25]. The bouncing effect of the ball is modelled fairly by both the constant velocity and constant acceleration models. The line at the bottom of the bouncing trajectories is the motion of the ball on the floor before coming to rest which both the models have captured.

With this sequence, it is worth observing the effect of changing learning rate from 0.001 to -1. As clearly visible in 7 for constant velocity model , there are noisy trajectories for learning rate = 0.001. In the beginning of the sequence, a blob gets detected (even in the absence of the ball) which is part of the wall. Specifically, illumination change caused the joint in the wall and floor to be detected as an object which lead to detection at those positions. Similar results are obtained for constant acceleration model. This has been resolved by decreasing the learning rate which helps to adapt to changes in the background more quickly. A similar reason lead to choosing of the learning rate for Video3 in the

previous case.

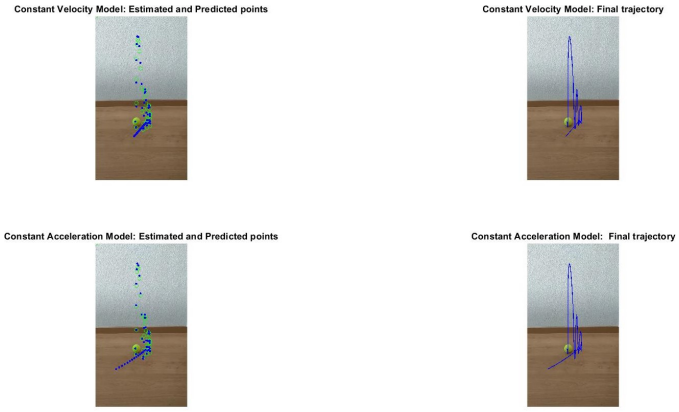

Fig. 6. Comparison of results obtained for constant velocity model and constant acceleration model(Video5.mp4). Blue circles: estimated points, Green circles: measurement points

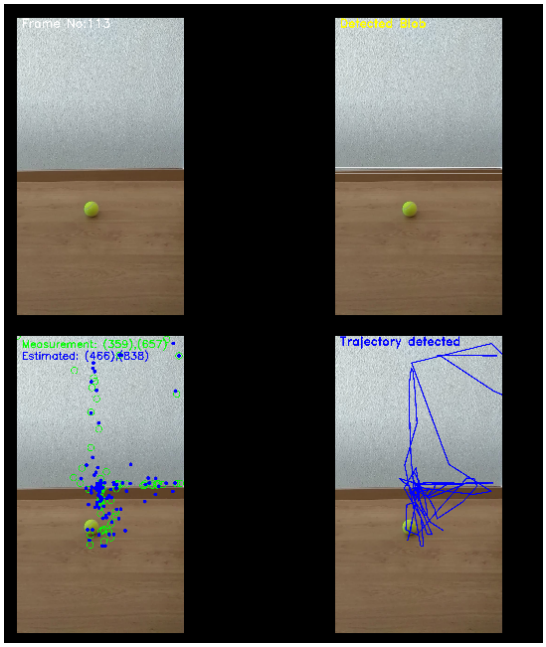

Fig. 7. Comparison of noisy result obtained at learning rate= 0.001 for constant velocity model (Video5.mp4). Blue circles: estimated points, Green circles: measurement points

Further, for the last sequence Video6, the parameters are set as: learning rate = 0.001, min_height = 50, min_width = 50, measurement noise covariance, R = [250,0;0 250]. Figure 8 shows the comparison of trajectories for constant velocity model and constant acceleration model from R = [0.25,0;0 0.25],R = [25,0;0 25],R = [250,0;0 250]. Similar to the case of Video3 in case of the occlusion the acceleration model with a smaller value for R drastically predicts a curved path for the ball. This happened when the tracker has a high belief on the prediction while updating the parameters. As observed by increasing the value of R we increase the noise when doing the update using the prediction which implies that the tracker would rely more on the measurements than the predictions. In the second row in Figure 8 the curvature of the predicted path decreases as we increase the value of R till it resembles the actual path the ball might have taken while being occluded. In comparison the velocity model represents the object path better. This happens because the motion of the object itself doesn't have a high acceleration and looks more uniform though at the end it hits the wall and stops. So the for the velocity model the changing of R does not produce as big an impact in changing the prediction as in the case of the acceleration model.

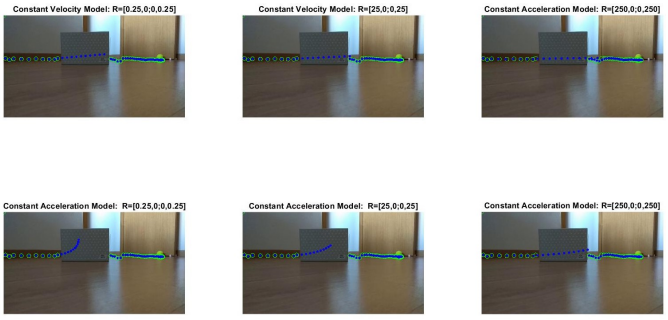

Fig. 8. Comparison of results obtained at learning rate= 0.001 for constant velocity model (Video6.mp4).Blue circles: estimated points, Green circles: measurement points

C. *TASK 3.3*

Each of the videos for the current task had different features and hence it was necessary to address them separately. The exact variations tried and the observations made have been briefly discussed in the following sections.

*1) **Abandoned Box**:* The min_height and min_width parameters for this sequence have to be set to (15,15) for the proper detection of the cyclist in the video. It was observed that having a learning rate of 0.0001 or smaller gave blobs for the cyclist that were oscillating between the upper body and the moving feet. So the blobs are detected as two boxes for the single cyclist. As the implementation considers the blob with the largest area, only one of the blob is considered and hence the centre moves up and down (slightly oscillating) which can be seen in the results in Figure 9.

Using a learning rate of -1 improves the detection of the blobs. The oscillating detection is reduced which results in much smoother path being tracked for the cyclist. In addition, towards the end when the cyclist becomes stationary the measurements stop and hence the velocity model predicts a few more points in a straight path. The acceleration model shows a curved path in the absence of the measurements.

*2) **Boats**:* Boats is the longest video in this task and it consists of a number of issues including a dynamic background of water, cars and walking people. In addition to this, the boat sails in a straight path for sometime and then takes a complete U-Turn which is where the trackers needs to be robust. As shown in Figure 10 the velocity and acceleration model both show large deviations when the boat

Fig. 9. Comparison of results obtained at learning rate= 0.0001 and final trajectory obtained with learning rate = -1 for constant velocity and acceleration models for Abandoned Box video. Blue circles: estimated points, Green circles: measurement points

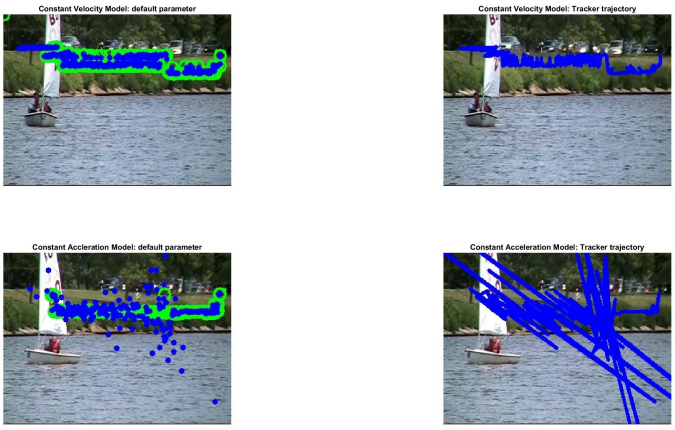

Fig. 10. Results for constant velocity and acceleration models using default parameters. In the left column Blue circles: estimated points, Green circles: measurement points. The blue lines in the right column are the trajectory of the estimated points

starts to take a turn. The velocity model predicts the boat on the other side and the acceleration model suffers from huge deviations.

It was observed that a lot of dense measurements were being made before the boat took a turn as it showed a small movement and the centers of the blob almost overlapped. Also when the boat was taking a turn there was a lot of measurements being missed. By decreasing the noise in the velocity component of the process covariance noise matrix (Q), the confidence of the tracker on the process (to better estimate the prediction) could be increased which lead to a reduction in the oscillations that were observed in Figure 10. Furthermore, once the boat takes a complete U-turn the filter wasn't able to track the straight path of the boat. The blobs that were being detected were small and had a high miss rate which reduced the tracker performance. By increasing the history parameter of the background subtraction model from 50 to 100, a significant improvement was observed as shown in Figure 11. Comparing this to the previous result, it can be safely concluded that the overall performance was

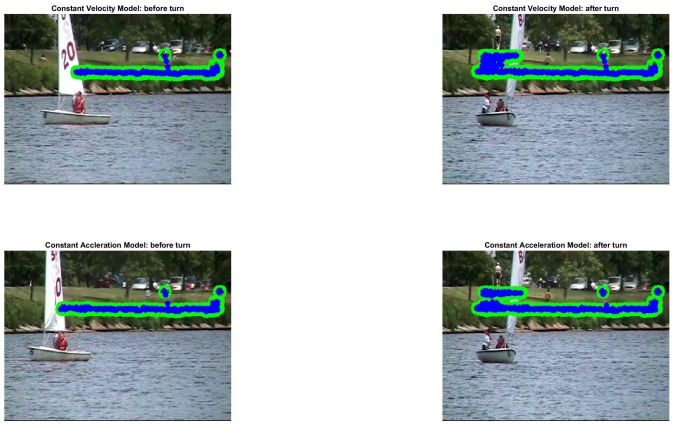

Fig. 11. Results for constant velocity and acceleration models using modified Q matrix and history = 100 (background subtraction). Blue circles: estimated points, Green circles: measurement points.

improved and the tracking system could handle the complex motion of the U-Turn of the boat. One significant issue towards the end of the video was that in the background two moving cars overlapped and got detected as a large blob which suddenly shifts the position of the estimated points. This particular issue couldn't be resolved just by handling the basic parameters by either of the models. Some form of comparing all the blobs detected in a frame and picking the blob closest to the one that is being tracked would be helpful in such cases. Also, some issues similar to that of the cyclist being detected in parts was observed with this sequence which lead to the center oscillating between the centers of the boat and sail separately. Trying to apply a larger morphological operation of closing was not successful in solving the oscillating sail to boat issue. But it was interesting to observe that the closing operation helped in removing the peaks that were a characteristic of both the models at the beginning of the tracking. The path produced after the closing also much smoother and uniform than before as shown in Figure 12.

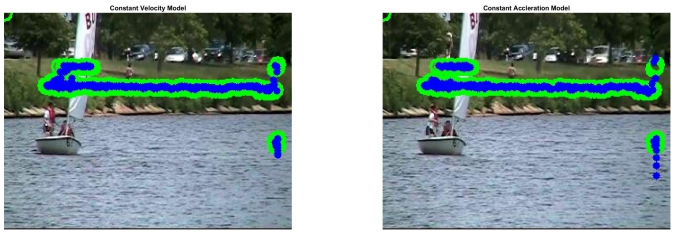

Fig. 12. Results for constant velocity and acceleration models after applying a closing morphological operation before extracting blobs. Blue circles: estimated points, Green circles: measurement points.

*3) **Pedestrians**:* The min_height and min_width parameters have to be set to (25,25) for the proper detection of the person walking in the video. The major challenge in this case was the shadow that was getting detected as blob with a larger area than the person which lead to bad tracking. One solution implemented for this was to increase the kernel size

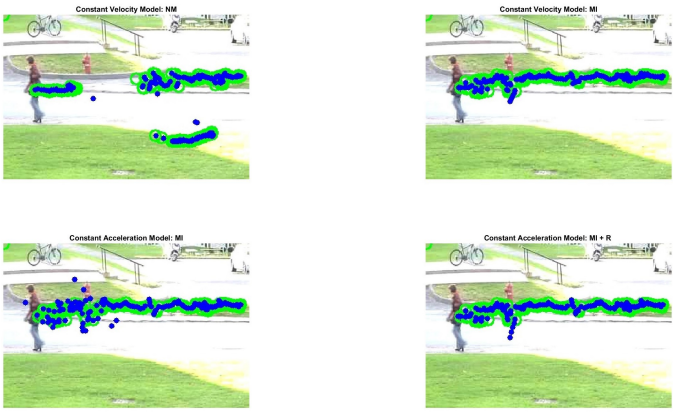

Fig. 13. Comparison of results of constant velocity and acceleration models on pedestrians video for different cases. NM: No Modification, MI: Morphological Improvement, MI + R: Morphological Improvement and change in R value of Kalman Filter. Blue circles: estimated points, Green circles: measurement points

of the morphological operation applied to the frames from (3,3) to be (7,7). This operation was successful in reducing the size of the shadow which then gave good results for the constant velocity model. Also, it can be seen that towards the end there are a lot of dense measurements that are made when the person stands near the spot for a few seconds. This does not affect the kalman tracker using constant velocity. The tracker correctly predicts the path of the object during the phase of the missed measurements.

While the morphological modification was sufficient for the velocity model, using the same parameters for the acceleration model caused large deviations in the trajectory when there were measurements made that we were very close to each other and were very dense as shown in Figure **??**. Trying different combinations of parameters it was observed that reducing the measurement noise covariance (R) values for the kalman filter reduced the amount of curve the model predicted for the path. The best parameter for R was found to be 20.0 for both the states. As presented before reducing the value of R implied to believe the prediction more than the measurement for this case (the person is stopping and moving again in intervals). There is a visible improvement in the performance as can be seen in the figure. The decreased R here reduces the values of updates for the filter which in turn does not let the trajectory deviate suddenly due to a large correction value.

*4) **Street Corner at night**:* The street corner video is a very short video of 3 seconds of a car whizzing past a signal at night. The main challenge here is as the car is at a high speed the tracker has to be accurate and due to the small area of the road the camera captures the car is visible only for a short time. The scene also consists of high intensity lights of the signals which could be detected as potential blobs when they blink. Without modifying the parameters the result obtained had many wrong measurements of the blobs of the traffic lights and also of nearby objects that got illuminated by the car light.

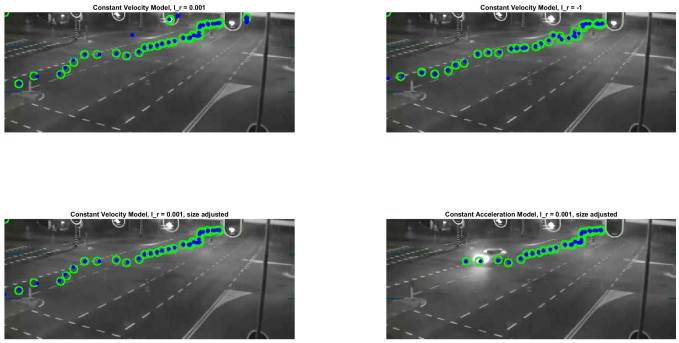

Fig. 14. Comparison of results of constant velocity and acceleration models on street corner video. l_r: learning rate, size adjusted implies the min_height and min_width parameter were set according to the video.

On trying a learning rate of -1, the results improved and a better trajectory was obtained. One major observation was that setting the min_height and min_width parameters to be (25,25) eliminated the other detections of traffic light and objects as they all were smaller than the car. In this case using a learning rate = 0.001 similar results to the one with learning rate = -1 was obtained as shown in Figure 14. This result was able to track the movement of the car through the frames to a relatively good accuracy. The same parameters produced good results with the acceleration model as well and didn't need to be modified.

VI. CONCLUSIONS

In conclusion, the implementation of the Kalman Filter has been validated on a toy dataset and then tested on two types of video sequence sets. The results from the evaluating on the different motions of the toy video itself proves that using such a tracker in controlled environment requires proper parametrization to be able to get proper results. Once the filter has been tuned to be working well in most controlled scenarios it is possible to apply it to real life video sequences. In the case of the Task 3.3 video sequences, the major challenge was the parametrization required to extract the measurements correctly. Also, it is evident that not all objects and motion can be represented by the same model correctly. Based on the situation under consideration a choice of a model has to be made.

Further, the measurements can be improved by improving the background subtraction models and considering some other parameters along with them. Also, instead of choosing only the largest blob in the scene a method can be developed to find the most similar blob detected and then use that for tracking. This would surely improve the results significantly. But such a design would have an added cost of computation per frame and so based on these points the performance of the current filter can be further improved.

VII. TIME LOG

- **Blob detection with Grassfire Algorithm**: 1.5 hour: 1 hour to set up the project, linking library. 0.5 hours for the implementation.

- **Blob tracking with Kalman Filter**: 3.5 hours:1 hour reading the slides and understanding the method. 2.5 hours for implementing, debugging and optimizing the implementation
- **Optimization**: 2 hours: Spent in removing unnecessary for loops and using matrix operations. Encapsulation using classes.
- **Evaluation**: 4 hours: For each task trying checking how it performs on each video. In addition, tuning the Kalman Filter parameters, learning rate and other parameters to tailor it for each sequence
- **Report** 6 hours: Writing, generating screenshots for figures, comparing on other dataset, proof-reading, editing and adding references.

REFERENCES

[1] OpenCV Docs: https://docs.opencv.org/3.4.4/

[2] Introduction to Image and Video Processing", Th.B. Moeslund, Springer, 2012

[3] The Kalman Filter: An algorithm for making sense of fused sensor insight